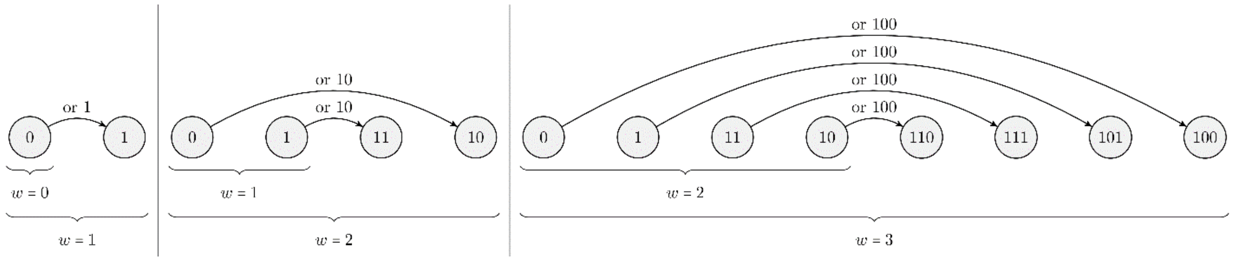
3. Casandra

Program Name: Casandra.java **Input File: casandra.dat**

Casandra's friend, Leah, recently discovered a way to efficiently determine the i^{th} Gray Code number. Leah observed that, for an arbitrary bit width w, that the Gray Code ordering for those 2^w numbers overlapped with the first 2^w numbers of the Gray Code ordering for width $w + 1$. The following is a visualization of this process starting from $w = 0$, to $w = 3$:



While this provides a nice visual understanding of this process, Casandra realized that the calculations shown above provides a decently slow process, as, to generate the i^{th} Gray Code, you must first generate the 0th through $i - 1^{\text{th}}$ Gray Codes. As a result, Casandra decided to investigate into methods of generating an arbitrary Gray Code number without the need to compute prior Gray Code numbers. In her research, Casandra found a process that allows her to further simplify this process using an XOR operation as follows, where g_i is the i^{th} Gray Code:

$$g_i = i \oplus (i \gg 1)$$

Having now discovered an efficient and independent way of generating Gray Code numbers, Casandra began to investigate some of the many applications of Gray Code numbers, one of which she found particularly useful: Karnaugh Maps. Karnaugh Maps can be used to efficiently and completely simplify Boolean expressions. For a given Boolean expression with V unique variables, first you must create a truth table for that Boolean expression, where you iterate over all 2^V permutations of true and false values. Suppose we had the Boolean expression:

$$Y = A \mid (B \,\&\, !C) \mid (C \,\&\, !B)$$

Then, the truth table for this expression would look like the following:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Next, a Karnaugh Map can be generated by simply re-arranging the values from the table above:

Y C \ AB	00	01	11	10
0	0	1	1	1
1	1	0	1	1

~ Casandra continued… ~

The process of re-arranging this table is as follows: First, given the number of unique variables, V, create a table of size $U \times D$, where $U = \left\lceil \frac{V}{2} \right\rceil$ ($V/2$ Rounded Up) and $D = \left\lfloor \frac{V}{2} \right\rfloor$ ($V/2$ Rounded Down). From there, the first U variables will be represented by the column headers, and the remaining D variables will be represented by the row headers (note that the variables in the truth table are sorted alphabetically, and thus, the order in the Karnaugh Map maintains that sorting). Next, generate the Gray Codes g_0 through g_{U-1}, and generate the Gray Codes g_0 through g_{D-1}. Take the Gray Codes g_0 through g_{U-1} and replace the column headers. Similarly, take the Gray Codes g_0 through g_{D-1} and replace the row headers. Lastly, fill in the table for values of Y corresponding to the values of the V unique variables from the original truth table. See below for how the Karnaugh Map above was derived.

Y	AB	g_0	…		g_{U-1}
C		00	01	11	10
g_0	0	$ABC = 000$ $\Rightarrow Y = 0$	$ABC = 010$ $\Rightarrow Y = 1$	$ABC = 110$ $\Rightarrow Y = 1$	$ABC = 100$ $\Rightarrow Y = 1$
g_{D-1}	1	$ABC = 001$ $\Rightarrow Y = 1$	$ABC = 011$ $\Rightarrow Y = 0$	$ABC = 111$ $\Rightarrow Y = 1$	$ABC = 101$ $\Rightarrow Y = 1$

Before learning how to use Karnaugh Maps to simplify Boolean expressions, Casandra wants to ensure that she understands the process of generating them perfectly. As a result, she decides to test her understanding by writing a program that, given a Boolean expression, generates an equivalent Karnaugh Map for that Boolean expression.

Input: The first line of input will consist of a single integer n ($1 \leq n \leq 50$) denoting the number of testcases to follow. The next n lines will each contain a single string denoting the Boolean expression that needs to be converted. Each of the n lines will consist solely of the characters [‘A’-‘Z’] (uppercase ‘A’ through uppercase ‘Z’), ‘!’ (the logical NOT operator), ‘|’ (the logical OR operator), ‘&’ (the logical AND operator), ‘^’ (the logical XOR operator), ‘(’ (the open parenthesis), and ‘)’ (the close parenthesis). It is guaranteed that no two variables are directly adjacent to one another in the string, and are separated by, at least, 1 non-variable character. Additionally, it is guaranteed that if character c appears in the list of unique variables, then all characters starting from ‘A’ leading up until c also appear. Lastly, $2 \leq V \leq 10$, as to prevent issues where the Karnaugh Map matrix is only a vector, or grows too large.

Output: For each of Casandra’s n requests, print out the $U \times D$ Karnaugh Map matrix of the Y-values, where each row is separated by a newline character, each column is separated with a single space, and each Karnaugh Map is followed by a newline containing the string “--------” (a string of eight hyphens).

Sample input:
```
4
A|(B&!C)|(C&!B)
A|B
A&(B|(C^!D))
!(A)|!(B)
```

Sample output:
```
0 1 1 1
1 0 1 1
--------
```
***Sample output continues next column*:**

***Sample output continued*:**
```
0 1
1 1s
--------
0 0 1 1
0 0 1 0
0 0 1 1
0 0 1 0
--------
1 1
1 0
--------
```